RasPi

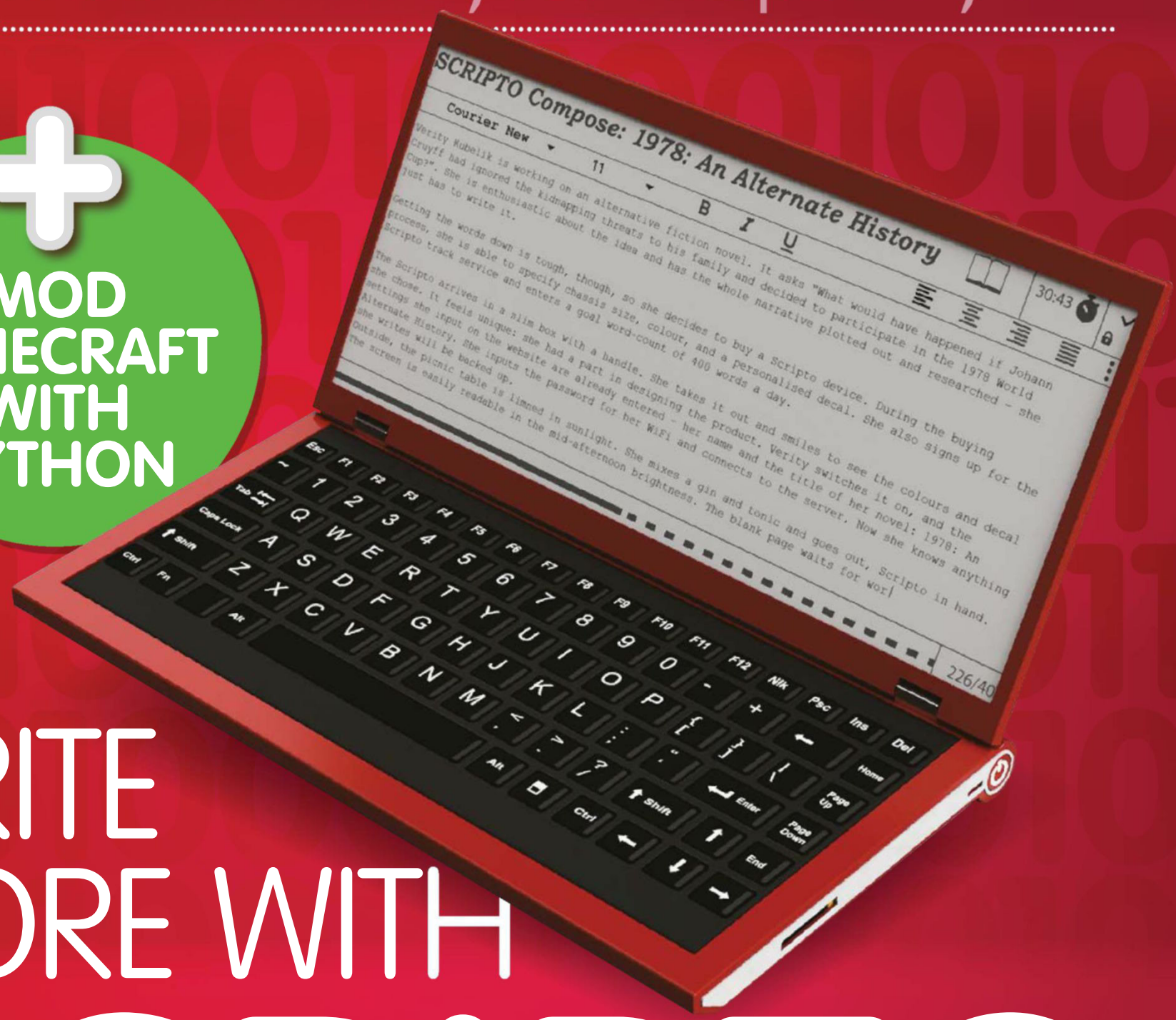
DESIGN
BUILD
CODE

44

Get hands-on with your Raspberry Pi

+

MOD MINECRAFT WITH PYTHON



WRITE MORE WITH SCRIPTO

Plus Turn the Pi into a remote hacking device

Welcome

Ever sat down to type up a report or work on a piece of code, then realised you've been looking at amusing animal videos on YouTube for the past 50 minutes? You're not alone. Although our computers are wonderful and capable powerhouses of productivity they're also packed full of distractions, from the wonders of the web to a simple solitaire session. The subject of our Pi-project showcase this issue is the Scripto, a distraction-free device that offers all of the functionality of a fully-fledged word processor with none of the attention sapping add-ons of a desktop or laptop. Swipe to the left a few times to find out how it was made...

Get inspired

Discover the RasPi community's best projects

Expert advice

Got a question? Get in touch and we'll give you a hand

Easy-to-follow guides

Learn to make and code gadgets with Raspberry Pi

Editor

From the makers of

LinuxUser & Developer

Join the conversation at...

@linuxusermag | Linux User & Developer | linuxuser@futurenet.com

Contents

Turn the Raspberry Pi into a remote hacking device

Using a few scripts, we're going to turn a Zero W into 'Rubber Ducky' pentesting tool

RubberDucky USB devices are great penetration-testing tools. This device is plugged into a target computer, and the USB drive tricks the computer into thinking it's a HID keyboard device in order to gain privileged access. Keyboards naturally provide a user with unrestricted access to the computer, in ways that a USB stick wouldn't normally be able to.

Pre-configured 'Ducky' scripts are then run on the target machine to prank the user or provide unauthorised remote access. Not only are we going to turn a Raspberry Pi Zero W into a USB device capable of running Ducky scripts, we're also going to gain remote access to the target machine in order to select which scripts we'd like to run, and gain shell access on the target PC.

For the sake of this tutorial we're assuming the target is running Windows and we - the attacker - are running a variant of Linux, but Rubber Duckys essentially work on any operating system. Scripts are available for Windows, Linux and OS X.

THE PROJECT ESSENTIALS

Raspbian Stretch Lite
www.raspberrypi.org/downloads

Etcher
https://etcher.io

N-O-D-E
https://github.com/N-O-D-E/Dong

RubberDucky Payloads
https://github.com/hak5darren/USB-Rubber-Ducky/wiki/Payloads

01 Preparation – the hardware

In order to get our Raspberry Pi set up as a USB device we'll need:

- A long USB cable with power adaptor
- A USB hub (for connecting multiple USB devices at the same time)
- A USB Ethernet adaptor and Ethernet cable (to gain internet access without having to mess around with Wi-Fi settings)
- A Mini HDMI to HDMI cable and a monitor to connect your Pi to
- A standard USB keyboard
- A microSD card

If you really want your Pi to look like a USB device, take a look at the N-O-D-E case (there's a link in the Resources section). Some soldering may be required. If you're not

using the N-O-D-E, you'll need a small USB to Micro-USB cable for connecting the Pi to your target PC.

02 Preparation - the software

Download the latest version of Raspbian Stretch Lite, and some software to write the image onto your microSD card – we recommend Etcher for this.

Once you've got Raspbian Stretch Lite installed, plug in a monitor and keyboard and boot your Pi. You can also use ssh for this step, if you can find the IP address of your Pi by checking your router or by using a network sniffer such as Angry IP Scanner. Once in, the default login details will be username: **pi** password: **raspberry**.

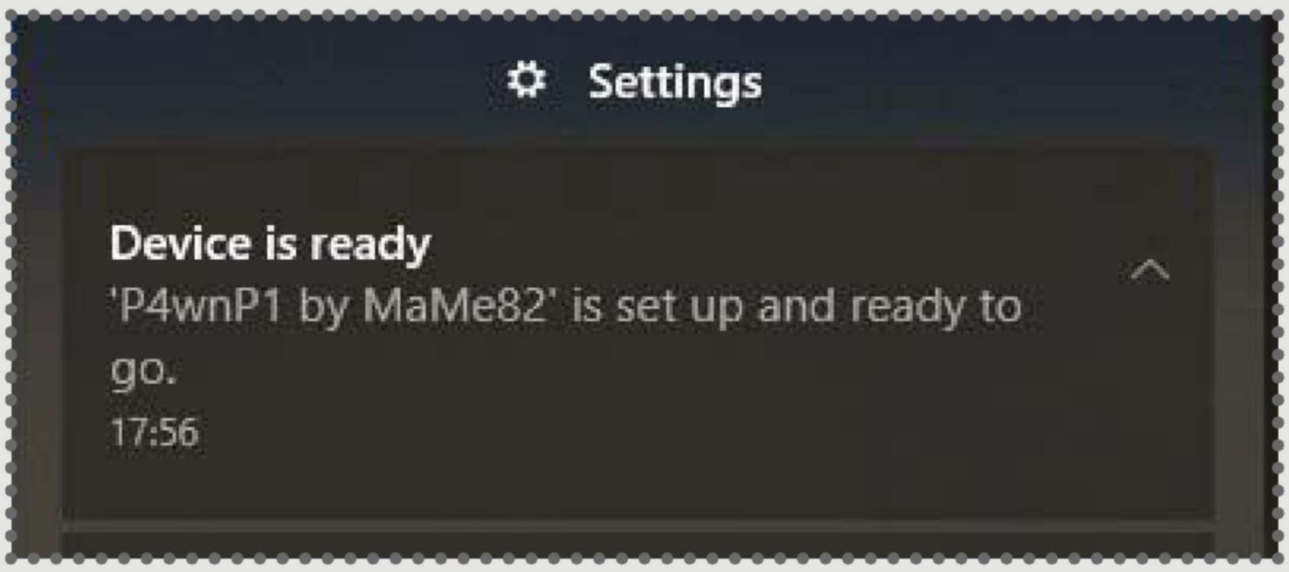
Next up we'll need to install git and download a clone of P4wnP1, which is the toolset that turns our Pi into a USB device.

03 Installation – git-cloning P4wnP1

Just run the following lines one by one:

```
mkdir ~/P4wnP1
cd ~/P4wnP1
sudo apt-get install git
git clone --recursive https://github.com/
mame82/P4wnP1
./install.sh
```

Grab a cup of tea, as installation may take some time. Once complete, note down the Wi-Fi name, key and SSH access displayed on the screen. We can of course change these later.



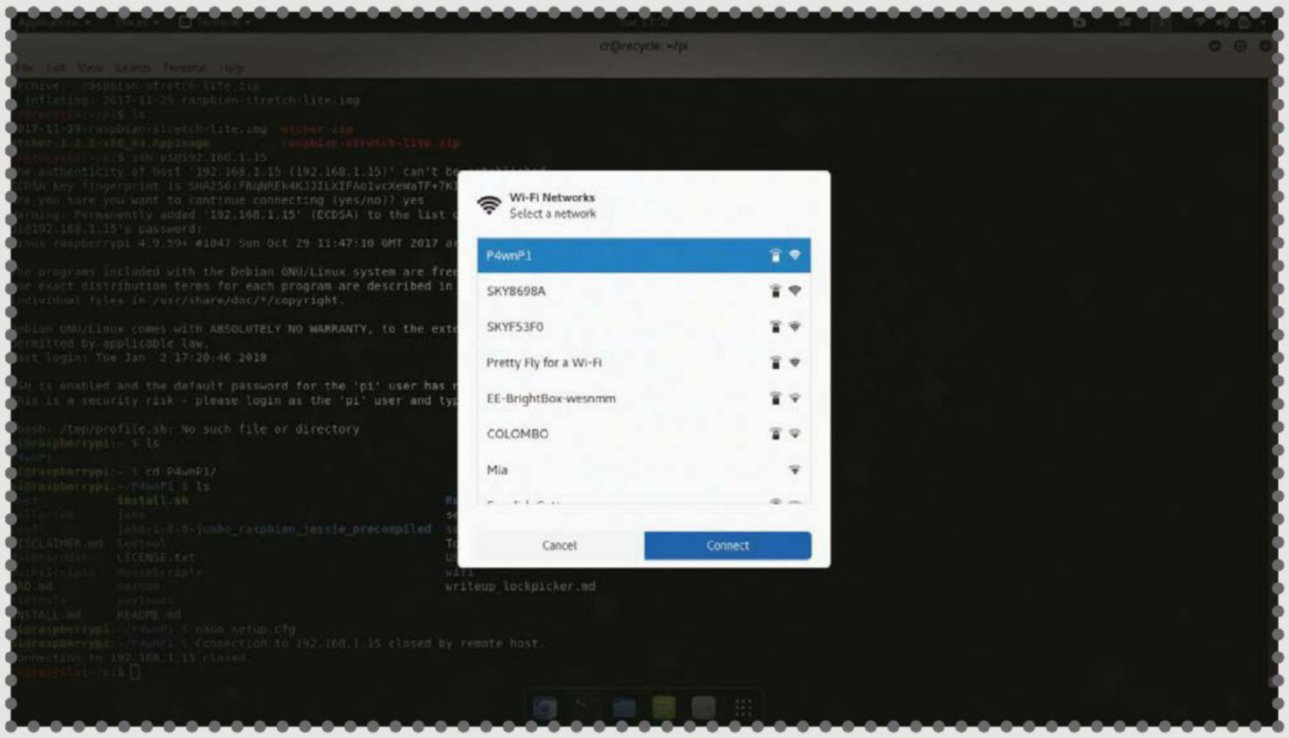
Where'd it go?

A really simple but fun prank is the old Alt-F4 script. This will press Alt-F4 on the target's screen, which in Windows immediately closes the currently active program, followed by Enter to bypass any 'Save' dialogue that may pop up to prevent the program from closing immediately. If you have line-of-sight this one can be a real treat, as you can run it every time the target re-opens the program.

```
ALT F4
DELAY 500
ENTER
```

04 Test the connection

Now that everything is set up, we should have a basic working P4wnP1 USB device. Before we set up our payload and customise our settings it's good to test that everything is working. We'll need two computers for this, one to be used as a target and the other for our remote control 'attacker'.



Plug the Pi into a target machine – which must be a working computer that is turned on – using the Pi's middle USB port (the one for data, not power). You should notice a couple of things: the target machine will display discrete pop-ups saying Setting up a device followed by Device is ready. At the moment, this new USB device will be called 'P4wnP1 by MaMe82' but we can change that later. On the attacker's machine we should see a new Wi-Fi network called P4wnP1, which means all is working as intended.

05 Customise your USB Pi

Now that the Pi is up and running, we'll want to either plug it back into a screen and keyboard, as we did earlier, or connect remotely over SSH at the address we noted down (172.24.0.1). Change directory into ~/P4wnP1 and run nano setup.cfg. Here you'll see a whole range

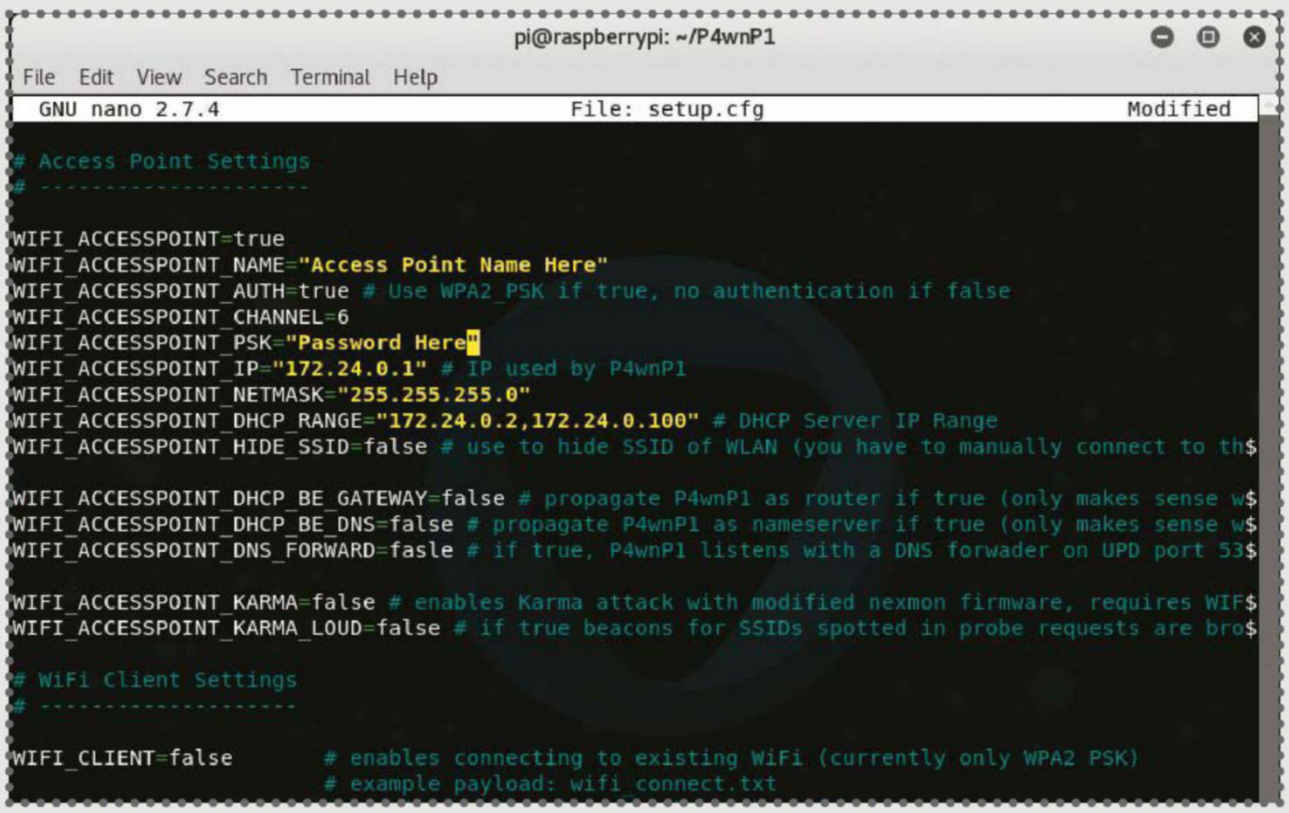
of settings, but ignore these for now as they'll mostly be overwritten by our payload config. What we want to do next is scroll to the end of the document and uncomment our payload of choice. For this tutorial we'll be using hid_backdoor_remote.txt, which enables all the fancy RubberDucky functionality. Be sure to comment out the network_only.txt payload with a #. Save and exit.

06 Setup your payload

Change directory to payloads and nano-edit the appropriate config file, in this case hid_backdoor_remote. Here you may want to change several settings, but most importantly WIFI_ACCESSPOINT_NAME and WIFI_ACCESSPOINT_PSK, which are of course the SSID and password required to remotely connect to your USB Pi. It may also be useful to change the keyboard language setting (lang) from us to gb.

"there are a number of other payloads to play around with in P4wnP1"

There are some rather interesting settings in this



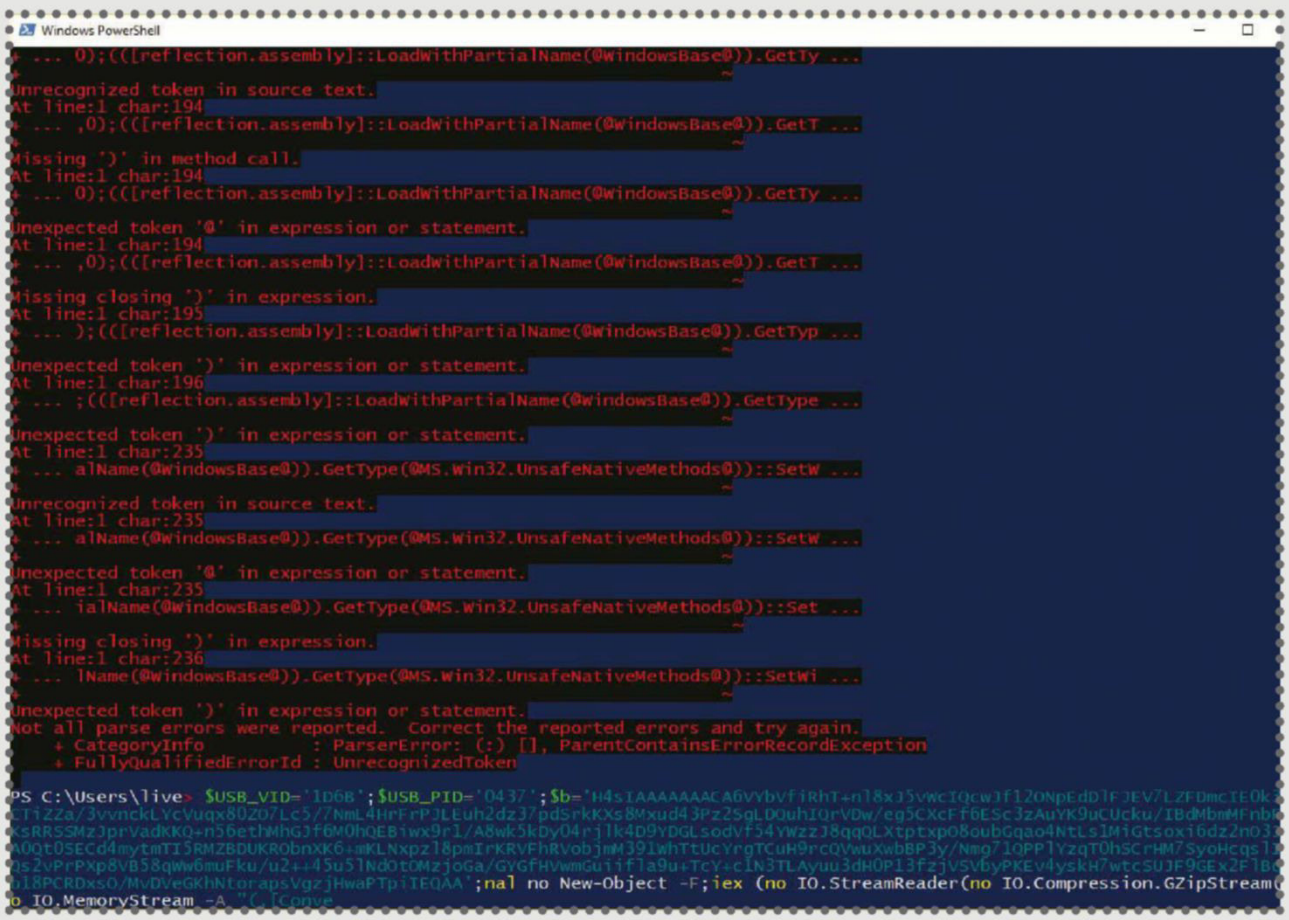
payload, namely the reachback connection or AutoSSH. This will enable the Pi device to automatically connect to a

a server of your choosing, via SSH, to essentially provide a backdoor tunnel.

07 Hack via Wi-Fi

While the AutoSSH functionality is fantastic, particularly for out-of-sight or long-range remote hacking, for the purposes of this tutorial we're going to stick with line-of-sight and/or short-range remote hacking via a local Wi-Fi connection.

Pop the Pi into a target machine and connect remotely via SSH to pi@172.24.0.1. A more discrete way of doing this, rather than using a laptop for attacking, could be to use an



Left The FireStage1 script running in PowerShell on target machine

Android mobile phone with a Terminal/SSH client installed. Once connected, type help for a list of commands. If you didn't change the keyboard layout in payload settings earlier you'll need to do so now, before passing any commands over to the target. GetKeyboardLayout shows the current setting and SetKeyboardLayout gives a list of

options.

08 Basic use

By default P4wnP1 shell will say client not connected. To gain remote access to the target machine we'll need to initiate the FireStage1 command. This will briefly open a PowerShell window on the target, before

taking advantage of a few exploits and disappearing again. We now have pretty much full control over the target machine – whatever the end user has access to, so do we.

Typing shell will give an MS-DOS-style command prompt, where you'll be able to use cmd as a regular user. Try running some basic Windows commands such as dir to see what happens. Type exit to quit the shell.

"We now have pretty much full control over the target machine..."

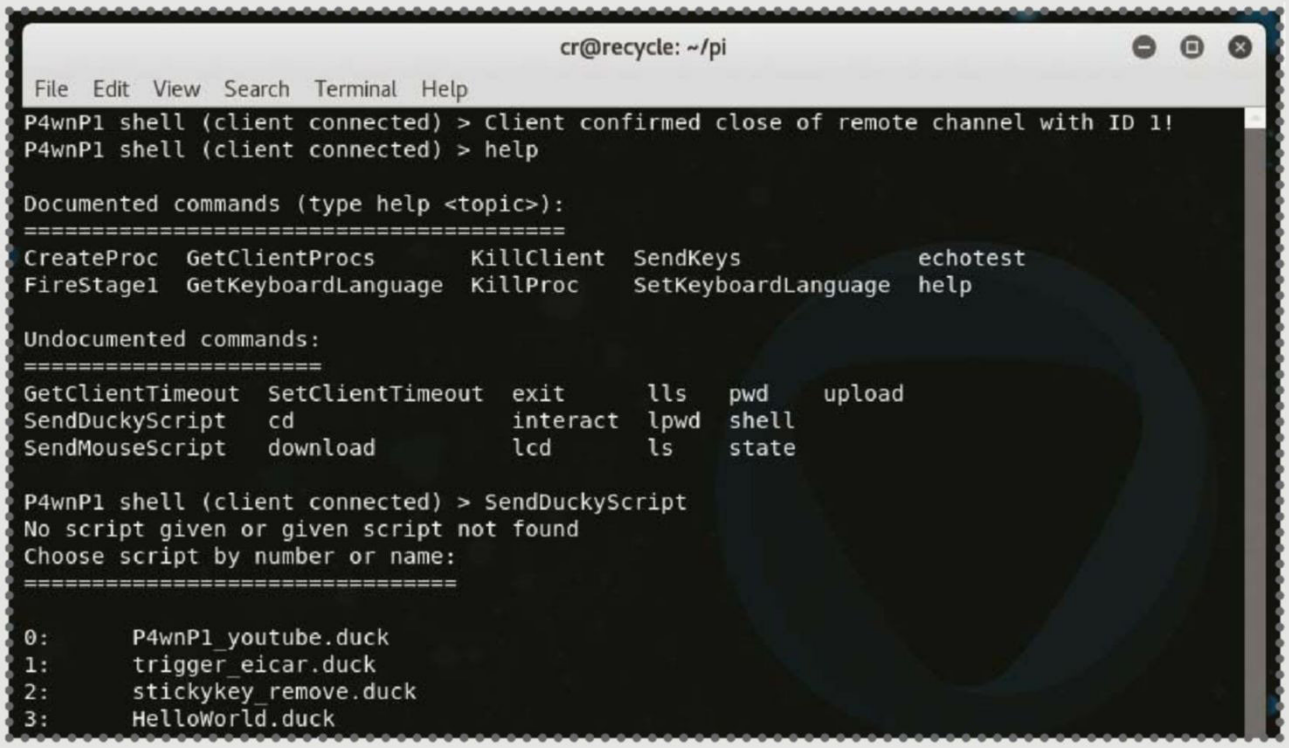
09 Playing with Rubber Ducky payloads

Having shell access is great, but we're really here for the RubberDucky scripts. To run one of these payloads simply type SendDuckyScript and you'll be greeted with a list of all the scripts currently stored on the microSD. By default there are seven scripts to play with, but there are also hundreds of other pre-configured scripts available online. We've linked to Daren Kitchen/Hak5's payloads in the Resources tab, where you'll find dozens of high-quality payloads.

P4wnP1-youtube.duck launches a YouTube video on the target machine, Trigger_eicar.duck checks for an installed antivirus. AltF4_Return.duck, AltF4.duck, Stickykey.duck and Stickykey_remove.duck are quite self-explanatory, while HelloWorld.duck opens a NotePad and types the message Hello World.

10 Edit RubberDucky payloads

We can open .duck files in a text editor such as nano and make our own customisations. HelloWorld.duck is a great place to start – try fiddling about with it. By default it looks like this:



```
GUI r
  DELAY 500
  STRING notepad.exe
  ENTER
  DELAY 1000
  STRING Hello World
  ENTER
```

11 Configure RubberDucky payloads

The delays are there to give the computer a chance to load software. GUI r opens Windows' Run dialogue window; our script then waits a few milliseconds before typing the string of text notepad.exe and pressing the Enter (Return) key. After another short delay (for Notepad to load) our script types out another string. We could of course edit this string or add multiple strings below it, to display our own custom messages on the target's screen:

```
  STRING Please remember to lock your PC and
protect your USB ports.
```

12 Add more RubberDucky payloads

By creating .duck text files in /P4wnP1/DuckyScripts we can collate as many RubberDucky scripts as we like, and they'll all be listed by the SendDuckyScript comment on our USB Pi.

```
  GUI r
  DELAY 500
  STRING iexplore -k http://fakeupdate.net/
win10u/index.html
```

A powerful weapon

Our Rasperry Pi Zero W is now an advanced Rubber Ducky USB device. We can take complete control of a remote machine, be it running Windows, Linux, Mac OS X or even Android. Remember to use this tool responsibly!

`ENTER`

This script loads a full-screen 'Windows Update' screen as a prank.

13 How to use hidden commands

There a few handy commands not listed under the help command, such as these:

`KillProc` Try to kill the given remote process

`KillClient` Try to kill the remote client

`CreateProc` This remote PowerShell method calls core_create_proc in order to create a remote process

`GetClientProcs` Print a list of processes managed by the remote client

`Interact` Interact with processes on the target. Usage: **Interact <process ID>**

`SendKeys` Print out everything on target through the HID keyboard

`exit` Exit the Backdoor payload and return to the Pi's command line

`state` See details about the target computer

echotest If the client is connected, command arguments given should be reflected back

14 Pi commands

P4wnP1 also allows for the use of some Linux commands, regardless of the target operating system:

- lcd Change directory on the Pi

`lpwd` Print the name of the Pi's current directory

`lls` Print the contents of the Pi's current directory

`pwd` Print the target's current directory

ls List contents of the target's current directory

`cd` Change the target's current directory

`upload` Upload a file from the Pi to the target. Usage: **upload <Pi/directory.filetype> <target/directory.filetype>**

`download` Download a file from the Pi to the target. Usage: **download <target/directory.filetype> <Pi/directory.filetype>**

`run_method` This is undocumented for now

15 Obtaining user credentials

Of course we've only used the network_only.txt and hid_backdoor_remote.txt P4wnP1 payloads in this tutorial, but others are available. Try switching to the hakin9_tutorial/payload.txt instead, as this takes things a step further.

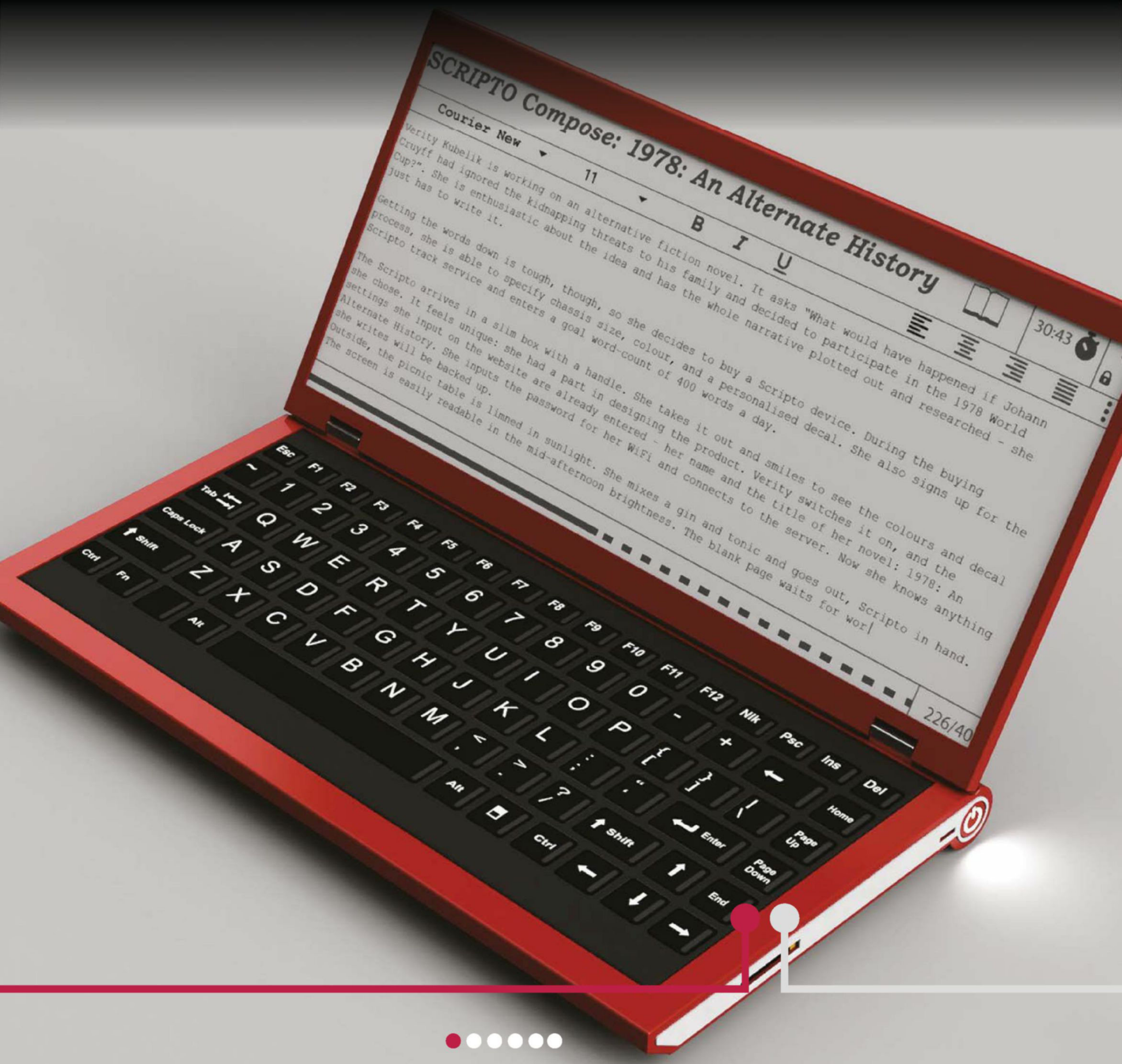
Instead of only replicating a HID keyboard interface, hakin9 also replicates a RNDIS network device and a USB mass-storage device. Therefore we can run a script that steals a user's credentials via PowerShell and then saves them directly to the USB. This would mean you could plug in the USB device, run the script, pull it out and walk away. The target would be none the wiser.

16 Other payloads

Once you're comfortable with hid_backdoor_remote and hakin9 there are a number of other payloads to play around with in P4wnP1. Win10_LockPicker attempts to grab Windows 10 login details, hid_mouse sets up the Pi to emulate mouse functionality instead of a keyboard, offering a completely different toolset, and wifi_connect is the infamous AuthSSH attack.

Write more with Scripto

Make procrastination a thing of the past with the help of Craig Lam's distraction-free writing tool



Where did the original idea for Scripto stem from?

The Scripto started as a final project as part of my Computing and Design degree at the Open University. The concept of a distraction-free device for writers is not new, but it is currently under-served in the market. The Raspberry Pi was also inspirational in providing an accessible computing unit for makers who aren't necessarily hardware experts.

Could you talk us through the design and build phase of Scripto? Did you encounter any major issues?

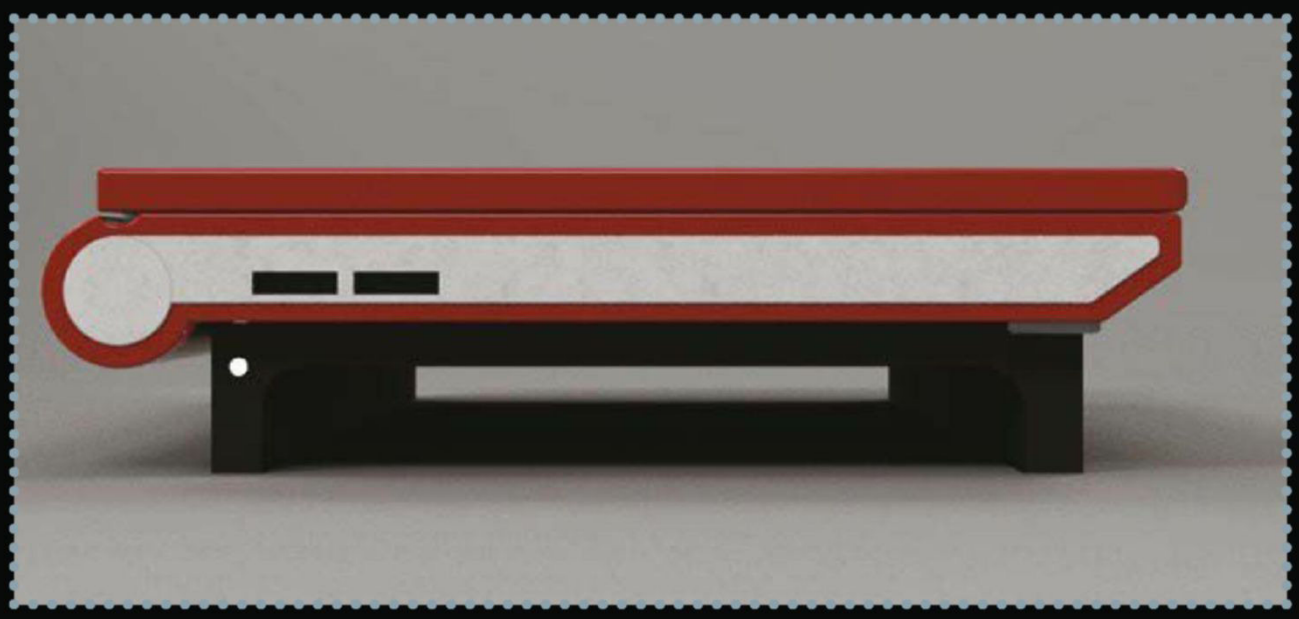
The Scripto was designed with three major goals in mind: it should be distraction-free, it should be convenient versus other solutions, and it should be cheap. Balancing the other factors with cost was a continual challenge. As the device is currently designed, I think the price would be too high and a major goal for the next few months is to redefine the budget to cut features that are nice-to-have but not strictly necessary, such as solar charging. For myself, using sustainable and innovative materials in the chassis was an important aspect of the design, but actual users may not care that their device is green. Another challenging aspect of the design is that the targeted demographic of 'writers' is extremely broad in terms of needs and disciplines. Script writers for theatre and film have different needs to poets and journalists. At a basic level, they all write words and should be able to replicate their chosen formats manually, however, making things convenient and

Craig Lam is a recent graduate, achieving First Class Honours in Computing and IT. His eight years of experience in CAD have helped him take Scripto from concept to creation.

seamless was a design goal, so the inclusion of templates and formatting requires an assessment of many competing formats. Every author seems to have a different workflow. It may be that the Scripto reverts to a pure text generation device, leaving it up to the writer to copy the basic text into their chosen format. Support for Markdown will be included at the least, however. A major challenge was deciding on the keyboard. Writers usually have preferences about their keyboard. Some prefer mechanical switches, others like the short-travel switches on modern Thinkpads and Macbooks. One thing is for sure: the keyboard must be no smaller than that found on 12" laptops. Writing is a physical act and getting the feel of the device correct will probably require many iterations. A challenge will be to find the resources and partners to help develop affordable prototypes.

For our readers who may not be familiar with Scripto, what are its key features?

The Scripto supports word processing with optional cloud backups. In essence: an old-school word processor like in the 80s and 90s, but with the convenience of automatic backups and progress



Left Each panel of Scripto can be customised. Different colours and materials are available to make it your own

THE PROJECT ESSENTIALS

- Raspberry Pi Zero (with enclosure)
- Biodegradable Arboblend chassis
- LCD touchscreen
- Sunpartner transparent solar panel
- Swappable side panels
- Wi-Fi antenna
- 4 x AA batteries
- SD card reader
- USB power input

tracking. It's a small, light device made of sustainable plastics using low powered hardware. A long-term goal would have it integrate with an app and a secure cloud storage service to track and backup users' words.

How does it deal with distractions?

The Scripto doesn't have any distractions: it should be a one purpose device. Beyond selecting which project you are writing in, the barriers between you and typing your thoughts should be minimal. Laptops, desktops, and smartphones are all capable of word processing, and they're also capable of running distraction-free environments – one could customise a basic Linux distro to boot into a word processor, for instance, but I think there is a benefit to having a very portable and dedicated device. Fostering a positive physio-psychological association with the act of writing is important, and specific hardware is a good way to achieve the mindset that "I'm sitting down with my Scripto to write, and nothing else".

Can Scripto be personalised in any way?

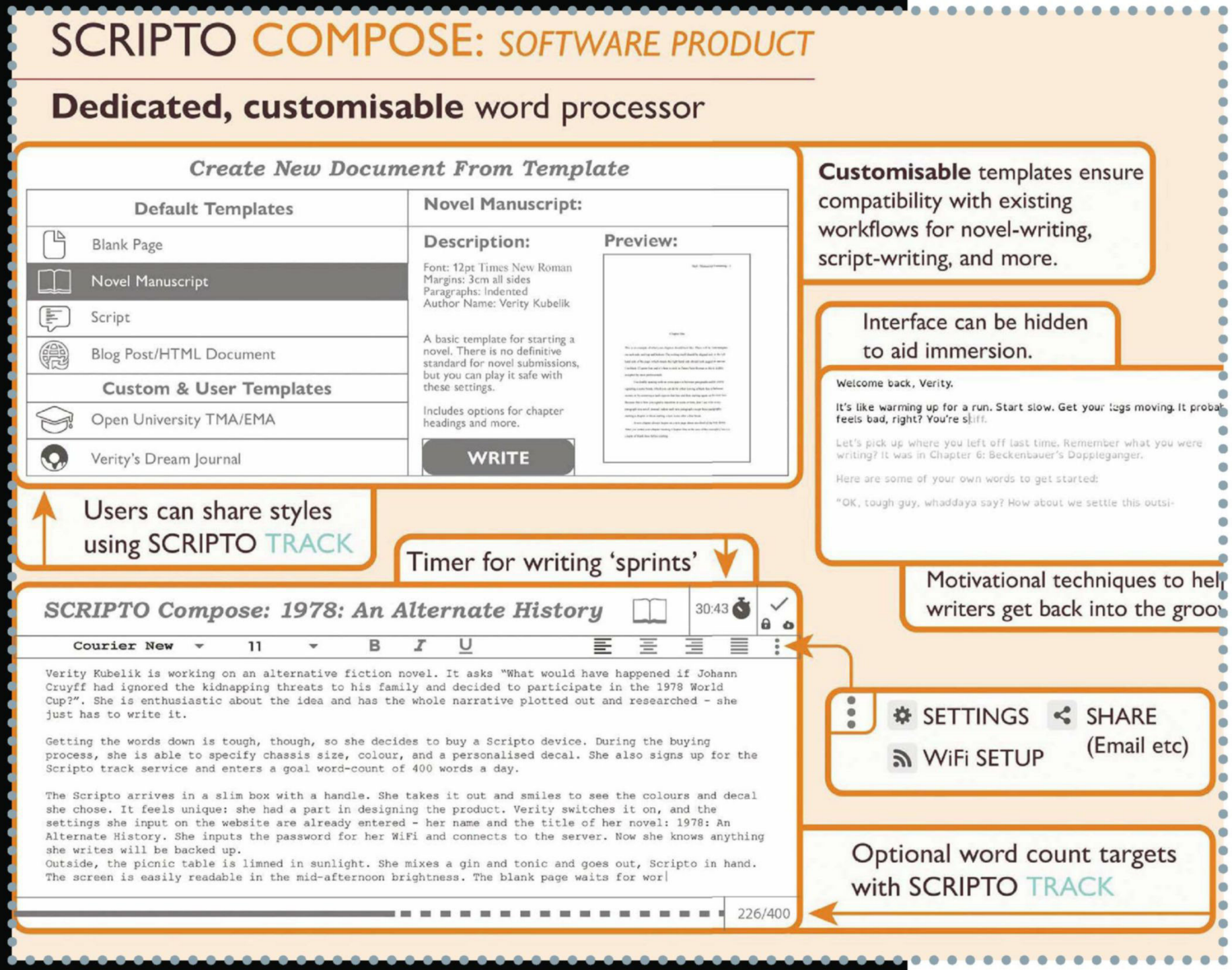
The intent is to make the Scripto a highly personal device. I go into tea shops and cafes and see rows of silver Macbooks – I have nothing against that, personally, but I think we're about to enter an age of mass customisation, and this should extend to our gadgets. Writing is individual and personal, so the tools we use should be as well. If the chassis can be 3D printed as intended, the long term goal is to

offer diverse colour options and decals. However, this becomes a challenge of pricing. Early versions may only come in one style and colour.

What sort of role does the Raspberry Pi play here? What were the benefits of using it in this project?

The Pi brings numerous benefits: primarily that it is a whole system on a single board and that it is compatible with existing drivers, software and accessories. The amount of documentation available is second-to-none. There may be more appropriate solutions – an Android based device may be better

Below While many core word processing features are present, the customised software used here is great for prolonging your focus



in terms of power management, but that requires research. At present, the project will use a Pi Zero. The Pi being UK-manufactured fits with the ethical and ecological aims of the project too.

Do you think you'll look to utilise Ras Pi in future projects?

I am always on the lookout for new technologies, but at the moment I could see myself using the Pi again. The Zero has an inspiring form factor – it could fit into all sorts of small interactive products; it's just finding time to explore!

What's next for you? Any other big projects on the horizon?

I recently graduated and I'm looking for work. The Scripto is an ongoing project, however, and I intend to have working prototypes available this year – my wife keeps nagging me! Crowdfunding may be an option if I wanted to scale up, but I'd prefer to have proof-of-concept before embarking on that journey. I also expect to be focusing on smaller projects, one of them being an online interactive guide to tea. I have a preoccupation with cats and so I'm investigating ways to make 'smart' cat toys.

Mod Minecraft on the Raspberry Pi using Python

Discover how to use Python to mod and tweak Minecraft

This tutorial is written with Minecraft Pi Edition in mind, but you don't have to be running Minecraft on a Raspberry Pi to follow along. We've put together a little package that will work on any version of Minecraft, so if you'd like to run this tutorial on your favourite flavour of desktop Linux – Pi or no Pi – you can. To allow Python to hook into Minecraft you'll need to install McPiFoMo by extracting the contents of the .minecraft directory into your ~/home/.minecraft. McPiFoMo includes MCPiPy from MCPiPy.com and Raspberry Jam, developed by Alexander Pruss. Provided you have Python installed, which of course is pretty standard on most distros, no additional software is required, other than your favourite text editor or Python IDLE.

Python scripts in this tutorial should always be saved in ~/home/.minecraft/mcpipy/ regardless of whether you're running Minecraft Pi Edition or Linux Minecraft. Be sure to run Minecraft with the 'Forge 1.8' profile, included in McPiFoMo, for your scripts to run correctly.

THE PROJECT ESSENTIALS

McPiFoMo
http://rogerthat.co.uk/McPiFoMo.rar

Block IDs
http://minecraft-ids.grahamedgecombe.com

01 Jump boost

Give your Minecraft player character a boosted jump by altering the Y factor to taste:

```
import mcpi.minecraft as minecraft
mc = minecraft.Minecraft.create()
playerPos = mc.player.getPos()
playerPos.y = playerPos.y + 10
mc.player.setPos(playerPos.x,playerPos.y,player
```

```
    Pos.z)
```

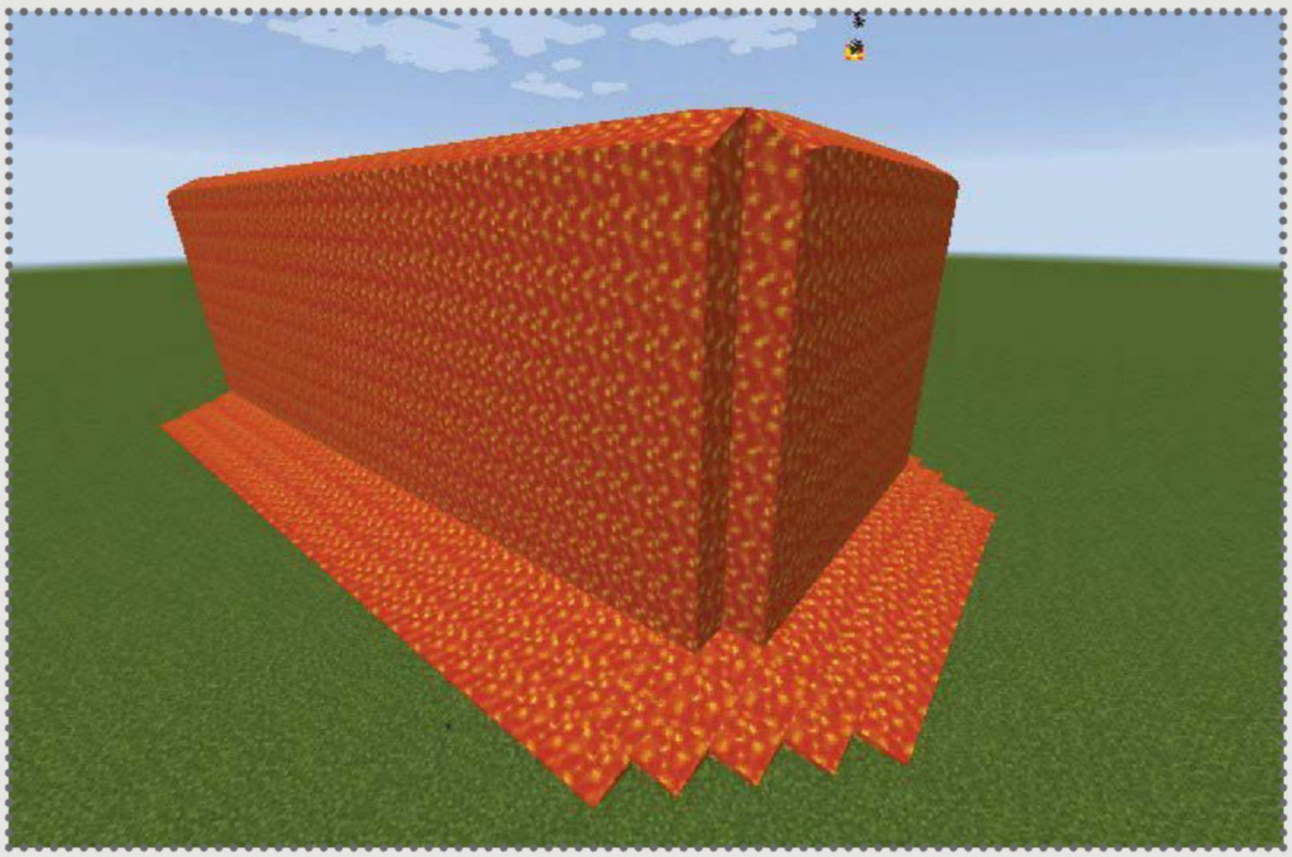
02 Auto-build

Give Minecraft a set of coordinates and the setBlocks command can fill in the gaps. Adding to the previous code:

```
  height = 10
  width = 10
  depth = 10
  blockID = 10
  mc.setBlocks(playerPos.x, playerPos.y,
    playerPos.z, playerPos.y + height,
    playerPos.x + width, playerPos.z + depth,
    blockID)
```

03 Setting a custom block ID for auto-build

Instead of hard-coding the blockID we can create a 'user input' to request one from the player. Working



with the code from steps 1 and 2, this method would mean commenting out the original blockID variable and replacing with the one below:

```
blockID = input("Enter your blockID: ")
  mc.setBlock(playerPos.x,playerPos.y,
    playerPos.z, blockID)
```

04 Activating Immutable mode

While we're experimenting with the auto-build tool we may want to stop ourselves and/or other players from destroying blocks. We can do this by setting Immutable mode, which makes blocks indestructible.

```
  mc.setting('world_immutable', True)
  mc.postToChat("Immutable mode activated")
```

You may also want to add an option to set this Boolean to False, to deactivate Immutable mode.

```
import mcpi.minecraft as minecraft
mc = minecraft.Minecraft.create()

mc.setting('world_immutable', False)
```

05 Glitchy signposts

For a glitchy sign, following on from the above code with playerPos and blockID variables initialised, you'll need to insert import time at the top of your code and then include:

```
states = [0, 5, 6, 11, 3, 8, 9, 2, 12, 7, 4,
  16]
```

```
    for state in states:
    mc.setBlock(playerPos.x, playerPos.y,
      playerPos.z, blockID, state)
    time.sleep(0.3)
```

06 Auto-foliage: making trees

Automatically building blocks is fine, but what if you want to automatically build an entire forest?

```
    def makeTree(x,y,z):
    wood = 10
    leaf = 10
    nothing = 0
    mc.setBlocks(x,y,z,x,y+3,z,wood)
    mc.setBlocks(x-2,y+4,z-2,x+2,y+5,z+2,leaf)
    mc.setBlocks(x-1,y+6,z-1,x+1,y+7,z+1,leaf)
    mc.setBlock(x-1,y+7,z-1,nothing)
    mc.setBlock(x+2,y+5,z-2,nothing)
    mc.setBlock(x+2,y+4,z+2,nothing)
    mc.setBlocks(x+1,y+6,z-1,x+1,y+7,z-1,nothing)
    mc.setBlocks(x+1,y+6,z+1,x+1,y+7,z+1,nothing)
    mc.setBlocks(x-1,y+6,z+1,x-1,y+7,z+1,nothing)
```

In this block of code we're simply creating a new function called makeTree with three passable integers for the coordinates.

07 Auto-foliage: spawning the trees

Now we need to spawn the trees. Change the integer to make them more/less dispersed:

```
    makeTree(x+1,y,z)
    makeTree(x+10,y,z)
```

"Automatically building blocks is fine, but what if you want to automatically build an entire forest?"

```
    makeTree(x+10,y,z)
    makeTree(x+1,y,z+10)
    makeTree(x+10,y,z+10)
    makeTree(x+10,y,z+10)
```

08 Hijacking another player's screen, part 1

List all the player IDs in chat:

```
    players = mc.getPlayerEntityIds()
    mc.postToChat(players)
```

Find the ID of the player whose screen you'd like to observe (or rather, hijack) and we'll use that number as an integer in the following step. Be sure to have permission from the player(s) whose screen you're about to take over beforehand, or there may be trouble…

09 Hijacking another player's screen, part 2

To take over another player's screen, we use the list created previously, and set our camera to follow that player:

```
    mc.camera.setFollow(players[1])
```

```
Switch back to your camera at any time with:
```

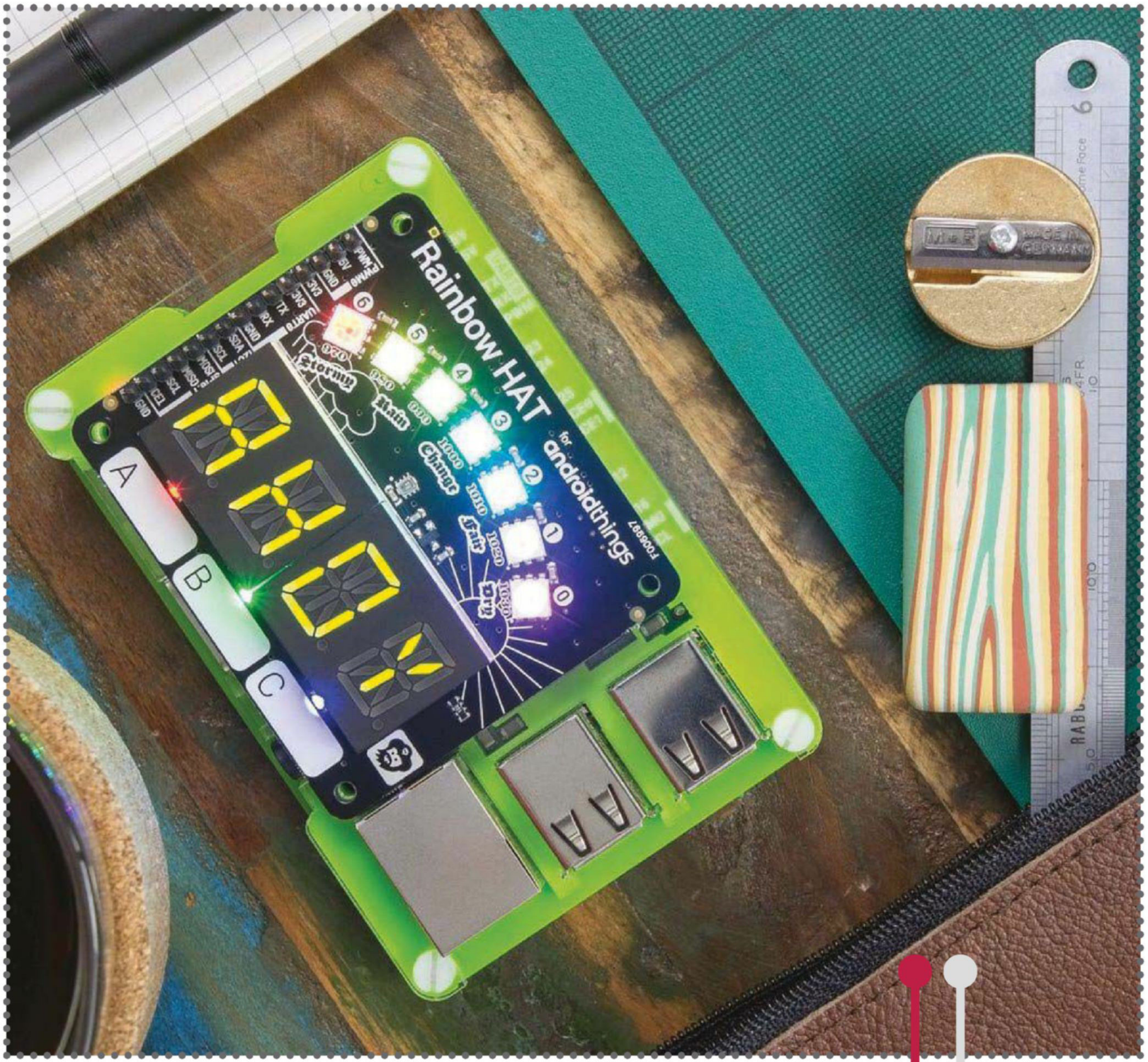
```
    mc.camera.setFollow(players[1])
```

Python for Minecraft Pi

Using Python we can hook directly into Minecraft Pi on the Raspberry Pi to perform complex calculations, alter the location of our player character, spawn blocks into the game world to create all kinds of creations – both 2D and 3D – and read/write pre-written scripts from text files to create our own authentic looking Non-Player Characters. We can program pretty much anything from pixel-art to chat scripts that communicate directly with the player.

Getting started with the Pimoroni Rainbow HAT

Combine LEDs, touch buttons and a sensor reading to create a real-time temperature display



Pimoroni is adept at creating awesome Hardware Attached on Top (HATs) for the Raspberry Pi. Last year it released the Rainbow HAT and, as expected, it's stacked with a buffet of sensors, inputs and displays to explore your surroundings. Use it as a weather station, a clock, a timer, a mood light or endless other things.

This tutorial walks you through some of the Rainbow HAT's main functions such as displaying text and taking a temperature reading. Then we move onto coding for the ten LEDs: everyone likes LEDs, and the Rainbow HAT boasts both mini-LEDs and seven full-RGB LEDs. Combine these with the built-in piezoelectric buzzer and the three touch buttons, and you have a simple all-in-one musical disco machine.

In the last steps of the tutorial, we'll create a real-time temperature display which changes and blends various shades of blue, orange and red as the temperature increases. It's a perfect little hack for monitoring when the warmth of spring approaches.

01 Installing the Rainbow HAT

With your Raspberry Pi turned off, attach the Rainbow HAT GPIO header to the GPIO pins – all HATs are designed to fit perfectly, so the hardware just slots into place. As with all Pimoroni products installation is clean and simple and a folder of example code and projects is included. Boot up your Pi, open the LXTerminal window and enter the code lines below one by one. On completion, restart your Pi.

```
sudo update
```

Hello world!

The Rainbow HAT uses 14-segment displays rather than 7-segment displays. This means that you can display proper text on them – upper and lower case as well as numbers and symbols. The buttons can be used to toggle things on or off and have a push-to-hold function that enables various different types of interaction.

```
sudo upgrade
sudo curl -sS https://get.pimoroni.com/
  Rainbow HAT | bash
```

02 Alphanumeric character display segments

To get started let's try a simple program to display some text. The four display blocks come in a range of colours and can be used to display the temperature, pressure or even the current time. Open your Python 3 editor and enter the program code below. This program imports the module, sets the text to display and then shows the text on the blocks. Try adding your own four-letter word (careful now…).

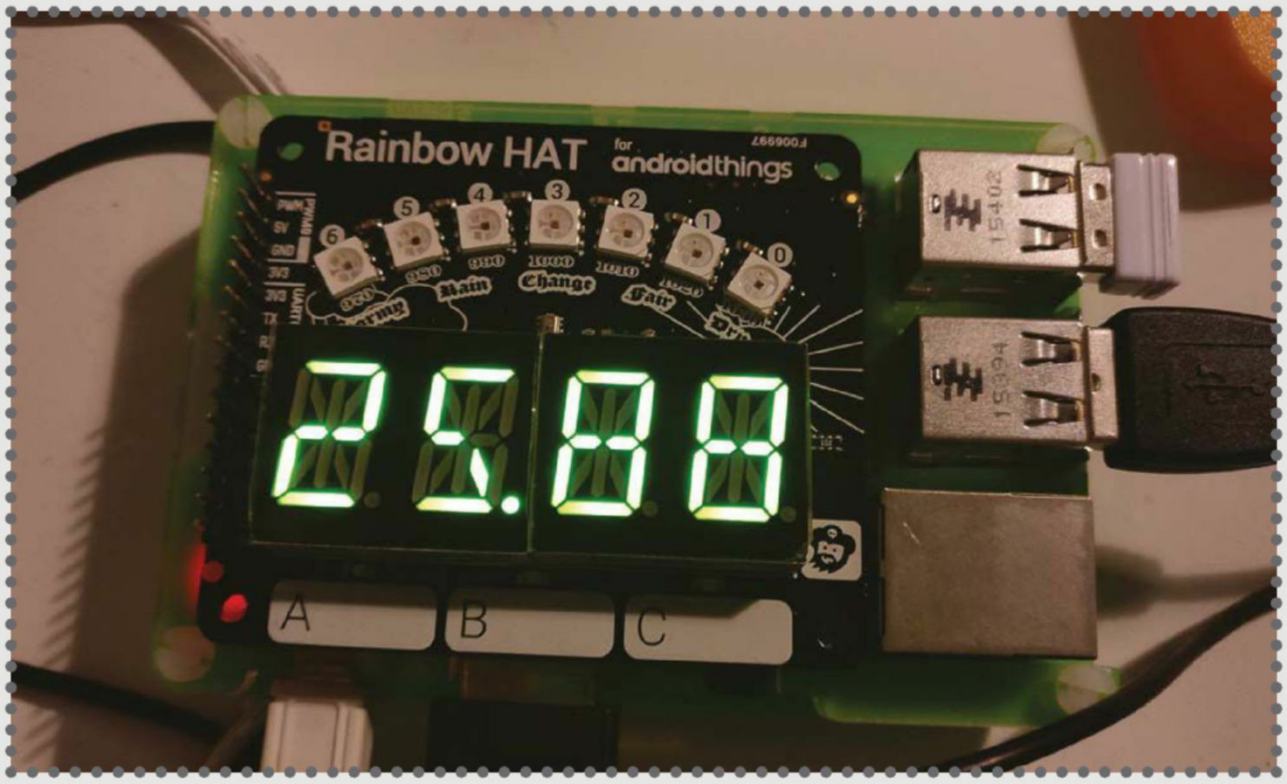
```
import rainbowhat
rainbowhat.display.print_str("LU&D")
rainbowhat.display.show()
```

03 The musical buzzer

The HAT comes with an inbuilt piezo buzzer which

can be coded to play different frequencies, producing different notes. The notes are based on standard MIDI values, where 60 is C at octave 5. The second value is the number of seconds that the note plays for. If you are adding more notes remember to add an equivalent delay to allow each individual note to play before moving onto the next one.

```
import rainbowhat
rainbowhat.buzzer.midi_note(60, 2)
```



Left The Rainbox Hat displays a temperature reading

04 Taking a temperature reading

This program demonstrates how simple it is to take a temperature reading and display it on the display blocks. First, we create a variable to store the temperature and use the code rainbowhat.weather.temperature() to take the reading, line 2. The reading needs to be converted into a float value before it can be displayed, line 3. Then display the reading on the blocks, line 4. You may notice that the temperature reading is slightly higher than the surrounding area. This is because the CPU is situated under the HAT and will obviously produce some residual heat.

```
import rainbowhat
temperature = rainbowhat.weather.
  temperature()
rainbowhat.display.print_float(temperature)
rainbowhat.display.show()
```

05 Mini-LEDs

The Rainbow HAT has three inbuilt mini-LEDs which can be used as indicators or status lights. The first LED is red, the second green and the third blue. These colours are preset and cannot be adjusted. To control the LED use

the code rainbowhat.lights.rgb(1,0,0) where the number 1 represents the LED being on and 0 off. The position of the each digit corresponds to the position of the LED. In the example code below, the first LED, red, is turned on.

```
import time
import rainbowhat
rainbowhat.lights.rgb(1,0,0)
rainbowhat.display.show()
```

06 Full LEDs

The seven main LEDs arc across the top of the HAT,

are bigger and can be adjusted for both colour and brightness. To turn on all the LEDs we use the code set.all, line 3. This is followed by four values; the first three values are the amount of red, green and blue, up to a maximum of 255. The last number sets the brightness level, where 1 is full and 0 is the lowest brightness. To turn off the LEDs we set all the colour values to zero, line six, and then show the LED, line seven.

Add the code below and experiment with the colours and brightness to see what effects you can get.

```
import time
import rainbowhat
rainbowhat.rainbow.set_all(100, 10, 100, 0.1)
rainbowhat.rainbow.show()
time.sleep(2)
rainbowhat.rainbow.set_all(0, 0, 0, 0.1)
rainbowhat.rainbow.show()
```

07 Individual LEDs

You can also program and control each LED individually. This is very useful as you have 16,581,375 available colours in total; combine this with the brightness settings and you have sufficient combinations to satisfy the needs of any project.

Individual LEDs use the code rainbowhat.rainbow.set_pixel followed by the LED position number, in this example 3. (This is physical LED number four as the numbering starts from zero.) The next three numbers correspond to the red, green and blue (RGB) values of the colours. The final number is the level of brightness.

```
rainbowhat.rainbow.set_pixel(3, 0, 255, 0,
0.1)
```

```
    rainbowhat.rainbow.show()
    time.sleep(2)
```

08 Touch button mini-LEDs

The Rainbow HAT boasts three touch-capacity buttons labelled A, B and C. They are controlled using the code touch.A.press() which will trigger an event on button A. In the example below, button A turns on the first mini-LED, denoted by the value (1,0,0) as used in line 5. Then you need to add the code to trigger the event on the button being released. In this example the value is set to (0,0,0) which turns the mini-LED off, line 8.

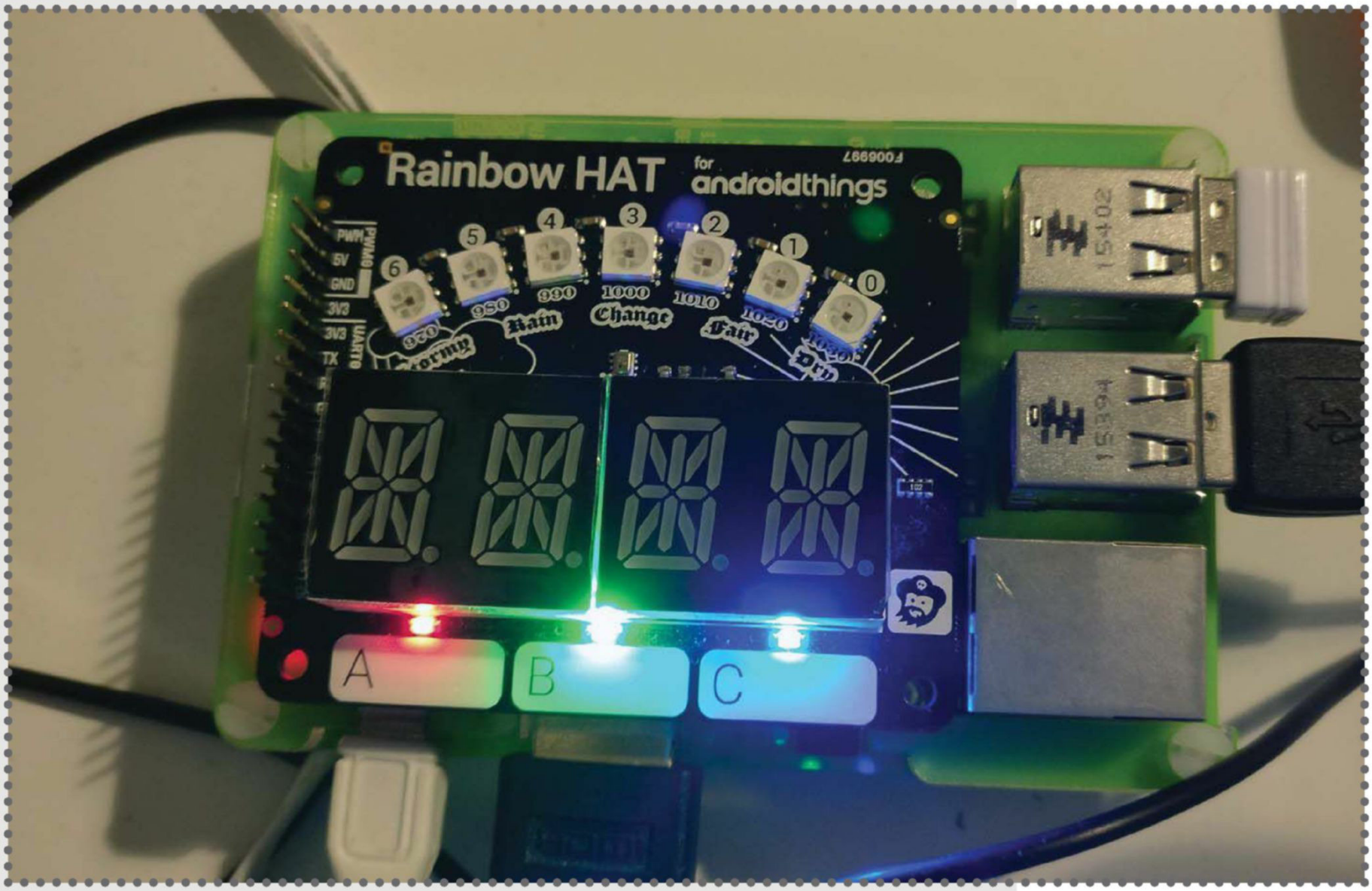
```
    import signal
    import rainbowhat

    @rainbowhat.touch.A.press()
    def touch_a(channel):
        rainbowhat.lights.rgb(1,0,0)

    @rainbowhat.touch.release()
    def release(channel):
        rainbowhat.lights.rgb(0,0,0)

    signal.pause()
```

09 Touch button LEDs

You can adapt the program used in step 8 to add code to trigger the large LEDs, line 6. This combines the code from Step 7 to turn on the third LED, the one in the middle. Remember that the colour of the full LEDs can be adjusted with the standard RGB values. Add the code to

Android Things

As well as using the RaspPi OS and Python to program your Rainbow HAT you can also use Android Things as Google has written drivers for the Rainbow HAT. To use them simply copy the Android Things for Raspberry Pi image to your microSD card, then install the Android SDK and you are ready to start developing your apps. You can check out more details of the project and example programs and code on Google's github site: https://github.com/androidthings/contrib-drivers/tree/master/rainbowhat.



respond to the release of the button, line 9, and you have a responsive LED at the touch of a button.

```
import signal
import rainbowhat

@rainbowhat.touch.A.press()
def touch_a(channel):
    rainbowhat.lights.rgb(1,0,0)
    rainbowhat.rainbow.set_pixel(3, 0, 255,
  0, 0.1)
    rainbowhat.rainbow.show()

@rainbowhat.touch.release()
def release(channel):
    rainbowhat.lights.rgb(0,0,0)
    rainbowhat.rainbow.set_pixel(3, 0, 0, 0,
```

```
    0.1)
        rainbowhat.rainbow.show()

  signal.pause()
```

10 Touch buttons buzzer – part 1

This little program combines the touch buttons, the mini-LEDs and MIDI notes to create a musical instrument! Begin by importing the signal and Rainbow HAT modules and then add the touch code for button A, line 3. Next create a function to respond to the button being touched. On line 5, add the line of code for setting the first mini LED to on (1, 0, 0) and on line 6, the code to play the musical note from the buzzer.

```
  import signal
  import rainbowhat

  @rainbowhat.touch.A.press()
  def touch_a(channel):
      rainbowhat.lights.rgb(1,0,0)
      rainbowhat.buzzer.midi_note(65, 1)
```

11 Touch buttons buzzer – part 2

Now add the code for Button B. Declare the touch.B.press() on line 1 and then create a function to store the lines of code to trigger the mini-LED and the buzzer. Change the rainbowhat.lights.rgb to (0,1,0,), line 3, to turn the middle LED on. Use the same code format to trigger events from button C by changing the touch.X.press(), and the rainbowhat.lights.rgb to (0,0,0).

```
  @rainbowhat.touch.B.press()
```

```
    def touch_b(channel):
        rainbowhat.lights.rgb(0,1,0)
        rainbowhat.buzzer.midi_note(80, 0.1)
```

12 Touch buttons buzzer – part 3

The final section of the program sets out what happens when the touch button is released. Create a function using the touch.release() code, line 1, and then set the RGB value to 0,0,0. This turns off all the mini-LED lights, notifying the user that the button has been released. Finally, add the signal.pauses() code to keep the program looping. Save the file, run it and create your musical `melody`.

```
@rainbowhat.touch.release()
def release(channel):
    rainbowhat.lights.rgb(0,0,0)

signal.pause()
```

13 Build a real-time temperature sensor – part 1

In this last project, we combine the Rainbow HAT hardware and sensors to create a real-time temperature display. Open a new Python file and import the required modules. On line 3, start a while loop and take the current temperature reading, storing it in a variable named temperature, line 4.

Convert the reading into a float value, line 5, and thendisplay the value on the display segments, line 6. Since the code is housed within a loop, the program will continually take a temperature reading and update the display accordingly.

```
import rainbowhat
```


```
import time

while True:
    temperature = rainbowhat.weather.
temperature()
    rainbowhat.display.print_
float(temperature)
    rainbowhat.display.show()
```

14 Building a real-time temperature sensor – part 2

In this step we need to compare the temperature to a preset value of 16. First we check if the temperature is less than 16 degrees Celsius (60.8 degrees Fahrenheit), line 1, and if it is, then we multiply the value by six, line 2. The product is stored in a variable named blue which is used on line 3 to set the LED colours. The colder it is, the less amount of blue is added to the RGB values.

```
    if temperature < 16:
        blue = temperature * 6
        rainbowhat.rainbow.set_all(0, 0,
blue, brightness=0.1)
        rainbowhat.rainbow.show()
```

15 Building a real-time temperature sensor – part 3

Now check for the high temperature value – it should be somewhere in the forties, line 1. This is very hot, so all green values are set to 0, line 2, and the LED RGB values are set to 255,0,0. This produces an intense red colour to represent the heat.

```
  elif temperature > 42 < 40:
            green = 0
            rainbowhat.rainbow.set_all(255,
    green, 0, brightness=0.1)
            rainbowhat.rainbow.show()
```

16 Building a real-time temperature sensor – part 4

If the temperature is between 17 and 39 degrees then the LEDs are coloured orange, increasing with intensity towards red as the temperature nears 40 degrees. A combination of red and green creates orange, so make a variable to store the amount of green as a product of the temperature, line 2. Then assign the values into the LED colours on line 4, before displaying them on the LEDs. Adjust the values to suit your climate, save the program and run it. You now have a real-time temperature display!

```
      else:
            green = 255 - temperature * 6
            print (green)
            rainbowhat.rainbow.set_all(255,
    green, 0, brightness=0.1)
            rainbowhat.rainbow.show()
```

Carry out portable geodesy

With the availability of Raspberry Pis, you can now do fairly complex geodesy calculations live in the field

The Raspberry Pi platform allows for projects where you can bring a computer with you when you go out into the field to do research. One of these research areas is geodesy (and geomatics), which is the branch of applied mathematics that's concerned with measuring and understanding the Earth's geometric shape. As you may have suspected, doing three-dimensional mathematics on a curved surface can be a tad messy. Hence the need to have some robust computing handy. The traditional way to do field work is to go out and collect measurements, then go back to the office. This month, we'll look at a Python module called PyGeodesy which can be used to handle these types of calculations. This isn't available in the regular package repos, so you will need to install it with sudo pip install pygeodesy.

One of the first things you may want to do is to convert data between different coordinate systems. There are separate objects that can be used to store data points within a particular coordinate system. For

example, you can use Universal Transverse Mercator (UTM) coordinates within your program and create an appropriate object with:

```
import pygeodesy
utm1 = pygeodesy.utm.parseUTM('31 N 448251
 5411932')
```

You can enter the data point as a string made up of the zone, hemisphere, easting and northing. The UTM object has properties which allow you pull out each of the individual elements, such as the easting, the northing or the hemisphere. If, instead of a UTM string, you have a set of latitude and longitude values, you can create a LatLon object and use that to create your UTM object. The following gives an example of how you could do this:

```
import pygeodesy
latlon1 = pygeodesy.ellipsoidalNvector.
 LatLon(49.66618, 3.45063)
utm1 = pygeodesy.utm.toUtm(latlon1)
```

This uses the usual ellipsoidal mathematics to make the calculations. There is also another set of equations, developed by Thaddeus Vincenty in 1975, that can also be used to manage points defined by latitude and longitude. The advantage to this alternative is that you can give it an Earth model to define the coordinate system by, rather than accepting the default ellipsoidal model. Once you have a UTM object, it has helper methods to convert it to other supported coordinate systems. The following code

gives a couple of examples:

```
# Convert to an ellipsoidal geodetic point
latlon2 = utm1.toLatLon(pygeodesy.
  ellipsoidalNvector.LatLon)
# Convert to an MGRS grid point
mgrs1 = utm1.toMgrs()
```

The first example enables you to convert back to a set of latitude and longitude values, but you need to provide which type of Earth model you wish to use by including the class of the type you are interested in. The second example converts the point to a NATO Military Grid Reference System (MGRS) grid point.

Now you have the ability to enter points, what can you do with them? One interesting problem is finding the central point given a series of geographical points on the globe. Again, you'd need to choose which ellipsoidal model that you wanted to use to do the calculations with. Using the defaults, you could do this:

```
import pygeodesy.ellipsoidalNvector as penv
mean_point = penv.meanOf(points=point_list,
  LatLon=penv.LatLon)
```

The variable point_list is a list made up of LatLon objects representing each point of interest. You also need to include what kind of ellipsoidal model is being used with the LatLon input parameter. Another task you may have with a series of points is to define a path along the surface of the Earth. Very often, you'll need to take measurements defining these points

Why Python?

It's the official language of the Raspberry Pi. Read the docs at python.org/doc

and simplify the path. PyGeodesy provides several simplification routines, of varied computational times, that provide different levels of simplification. Below is a basic simplification routine:

```
import pygeodesy.simplify as psimp
 simplified_path = psimp.simplify1(latlon_list,
  dist)
```

The first input parameter is the original list of points, given as a list of LatLon objects. The second input parameter is a tolerance distance. Any line segments below this threshold get removed, and the list of simplified LatLon points is returned. This is probably the fastest, and most inaccurate, simplification routine. Luckily, there are six other routines available, e.g. the code below uses the Reumann-Witkam algorithm:

```
 simplified_path = psimp.simplifyRW(latlon_
  list, pipe)
```

The first parameter is the original list of points to be simplified. The second parameter is the radius, in metres, of an imaginary pipe. This pipe is passed over the points in the original path, and all of the points lying within the pipe get simplified to a single line, up to the first point that lies outside the bounds of the pipe. An even more complex algorithm is the Visvalingam-Whyatt (VW) method of simplifying a path. This method creates triangles out of nearby points and tries to remove any external points if the triangle made is below some threshold. For example:

```
simplified_path = psimp.simplifyVW(latlon_
  list, area)
```

The second parameter provides the threshold area of the triangle, given in square metres.

Until now, we have looked at the ellipsoidal sub-modules, which model the Earth as some form of an ellipsoidal. There are also two sub-modules which model the Earth as a sphere: sphericalTrigonometry and sphericalNvector. Both of these sub-modules have their own versions of the LatLon class, and a whole set of module functions. The trigonometric version creates a new LatLon object, achieved by simply giving the values for the latitude and longitude. The newly created object has several instance methods available. Below are a few examples:

> "Doing 3D mathematics on a curved surface can be messy"

```
import pygeodesy.sphericalTrigonometry as
  ptrig
latlon1 = ptrig.LatLon(45.00, 66.00)
# Calculate a destination point
latlon2 = latlon1.destination(dist, bearing)
# Calculate the midpoint
midpoint = latlon1.intermediateTo(latlon2,
  0.5)
```

The first example takes a given point and, using a given distance and bearing, calculates what the resulting point would be. The second example takes two points and returns the point that is some ratio between the two ends. In this example, we are looking at a point halfway between the two ends. Instead of defining single points or lines, you can also

define a polygon by creating a list of LatLon objects. You can find the area of such a polygon, bounded by the great circles defined by these points, with area = ptrig.areaOf(latlon.list). This area is given in square metres. You may want to know whether one of the poles is enclosed by this polygon that you have constructed. You could find out with:

```
if ptrig.isPoleEnclosedBy(latlon_list):
    print('There is a pole here')
```

If there is no pole enclosed, you may be interested in finding the geometric mean of this polygon with mean_point = ptrig.meanOf(latlon_list). All of these are also available within the Nvector version, as well. This version uses N-vectors to define points and do the calculations. These underlying calculations are much easier to do and understand, rather than the trigonometric versions.

We've not mentioned other methods that can be used, such as trilaterate, but as you can see, there's lots of portable computing power that you can bring to bear.

Next issue

Get inspired **Expert advice** **Easy-to-follow guides**

"Control your house from your Pi"

Get this issue's source code at:

www.linuxuser.co.uk/raspicode